

# 3D Portfolio – Server

## DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

Höheren Abteilung für IT-Medientechnik

Eingereicht von:

Ema Halilovic

Betreuer:

Patricia Engleitner

Natascha Rammelmüller

Leonding, September 2023

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, Dezember 2023

Ema Halilovic

# Abstract

*3D Portfolio Gallery – Server* is the back end of the application 3D Portfolio Gallery, which was developed by Litzlbauer Lorenz and Maar Fabian. Thanks to it, designers get the possibility to upload their work online, where it gets displayed in a three-dimensional room.

The server is developed by Halilovic Ema and provides an interface to provide the front end with all the needed data. This data is saved in a database, which also enables file upload. The programmed interfaces are secured by a token system.

The technology used to build the server was Quarkus, along with a PostgreSQL database. For token management, JSON Web Tokens were used.



# Zusammenfassung

*3D Portfolio Gallery – Server* ist das Backend der Applikation 3D Portfolio Gallery, welche von Litzlbauer Lorenz und Maar Fabian programmiert wurde. Sie bietet Designenden eine Möglichkeit, eigene Werke innerhalb eines dreidimensionalen Raumes auszustellen.

Der Server wurde von Halilovic Ema erstellt und bietet der vorher erwähnten Arbeit Schnittstellen, um alle Daten abrufen zu können. Die dafür benötigten Daten sind auf einer Datenbank gesichert. Dank dieser wird ebenso das Hochladen von Dateien ermöglicht. Die ausprogrammierten Schnittstellen werden durch ein Token-System verschlüsselt.

Die, für den Aufbau des Servers verwendete Technologie, ist Quarkus, gemeinsam mit einer PostgreSQL-Datenbank. Für die Token-Verwaltung werden JSON Web Token verwendet.



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Ausgangssituation . . . . .	1
1.2	Zielsetzung . . . . .	1
<b>2</b>	<b>Technologien</b>	<b>2</b>
2.1	Quarkus . . . . .	2
2.2	PostgreSQL . . . . .	4
2.3	IntelliJ IDEA . . . . .	4
2.4	LeoCloud . . . . .	5
2.5	Notion . . . . .	5
2.6	PlantUML . . . . .	6
2.7	JSON Web Token . . . . .	6
<b>3</b>	<b>Umsetzung</b>	<b>7</b>
3.1	Design und Planung . . . . .	7
3.2	Aufsetzen der Datenbank und des Servers . . . . .	7
3.3	Hosten auf einer Cloud . . . . .	15
3.4	Schreiben der Tests . . . . .	15
<b>4</b>	<b>Zusammenfassung</b>	<b>16</b>
	<b>Glossar</b>	<b>V</b>
	<b>Literaturverzeichnis</b>	<b>VII</b>
	<b>Abbildungsverzeichnis</b>	<b>IX</b>
	<b>Tabellenverzeichnis</b>	<b>X</b>
	<b>Quellcodeverzeichnis</b>	<b>XI</b>



# 1 Einleitung

## 1.1 Ausgangssituation

Als Ausgangsbasis wurde die Diplomarbeit '3D Portfolio Gallery' hergenommen und mit folgenden Funktionalitäten erweitert:

- Hosten der Webseite innerhalb einer Cloud
- Implementierung von REST-Schnittstellen

Die genannte Arbeit, erstellt von Litzlbauer Lorenz und Maar Fabian ermöglicht die Erstellung von interaktiven und dreidimensionalen Räumen. Diese Räume werden mittels Angular in einem Browserfenster abgebildet.

Die Bereitstellung der benötigten Daten in Form eines Backends, ist die Hauptbeschäftigung dieser Diplomarbeit.

## 1.2 Zielsetzung

Das Hauptziel dieser Arbeit ist es, eine API zu erstellen, welche die Kommunikation zu serverseitigen Daten ermöglicht. Dafür ist eine Abbildung der Datenstruktur in einer Datenbank erforderlich. Damit diese fehlerfrei erfolgt, wird ein ERD verwendet.

## 2 Technologien

Für den serverseitigen Bereich gab es folgende Kriterien, welche von den benutzten Technologien getroffen werden mussten:

- Sie sollen auf dem neuesten Stand der Technik sein
- Über eine ausreichende Dokumentation mit ständiger Weiterentwicklung verfügen
- Eine weite Auswahl an Frameworks und Funktionalitäten unterstützen in den Bereichen:
  - REST
  - Filemanagement
  - Datenbank

listin gwas die Frameworks machen können in welchem bereich

Innerhalb dieser Arbeit kamen einige Technologien in Frage. Allerdings wurden die endgültigen Technologien aufgrund schon vorhandene Praxiserfahrung getroffen.

### 2.1 Quarkus

Das open-source Framework Quarkus, wird verwendet um cloud-native Projekte in Java zu entwickeln. Vorteile dieses Frameworks sind die kurzen Startzeiten, sowie der geringe Arbeitsspeicherverbrauch. [1]

Nach der Erstellung eines neuen Projekts wird standardmäßig eine Maven-Struktur erstellt, sowie Quarkus-Dateien, welche die Projekteinstellungen modifizieren können, wie zum Beispiel die *application.properties*-Datei. Zusätzlich verfügt Quarkus über eine Vielzahl von Extensions, welche durch Command-line-Befehle oder händisch zu Projekten hinzugefügt werden können. Um dies und weitere Quarkus-Aktionen zu vereinfachen, bietet dieses Framework ein zusätzliches Quarkus-CLI. [2, 3]



### 2.1.1 Maven

Da die Kompilierungsprozess eines Projektes recht komplex werden können, wird Maven verwendet, um diese zu vereinfachen. Ein gutes Beispiel eines komplexen Kompilierungsprozess ist die Ausführung eines Projekts auf unterschiedlichen Geräten mit verschiedener Hardware und Konfigurationen. Durch die Verwendung von Maven wird garantiert, dass dieses Problem nicht auftritt, da die einzige Voraussetzung des Zielgerätes nun ist, dass Maven installiert und eingerichtet ist.

Mit einem "Maven-Ordner" können Projekte mit gewohnten Maven-Befehlen ausgeführt werden, ohne dass eine Installation des Tools notwendig ist. Quarkus-Projekte verwenden von Mavens *pom.xml*-Datei, um zum Beispiel die verwendete Java Version oder alle verwendeten Extensions abzuspeichert. Zusätzlich ist es möglich, ein einheitliches System für Projektkonfigurationen zu bieten. Dadurch müssen Einstellungen nicht mehr manuell bei Gerätewechsel getroffen werden. [4]

### 2.1.2 JDBC Driver - PostgreSQL

Für Quarkus Projekte gibt es eine Extension namens "*JDBC Driver - PostgreSQL*". Diese ermöglicht eine Verbindung zu PostgreSQL-Datenbanken. In Java versteht man unter der JDBC eine API für Java-Anwendungen. Die JDBC-Driver sind Implementationen der API für den benötigten Fall, wie hier für PostgreSQL [5].

Um die Extension verwenden zu können und eine Datenbankverbindung aufzubauen, müssen in den *application.properties* einige zusätzlichen Konfigurationen eingefügt werden. Wichtig sind Informationen, wie die Art der Datenbank, der Pfad, um diese zu erreichen, und die Login-Daten eines berechtigten Nutzers 1:

Listing 1: Beispielkonfigurationen

```
1 quarkus.datasource.db-kind=postgresql
2 quarkus.datasource.username=meinUser
3 quarkus.datasource.password=meinPassword
4 quarkus.datasource.jdbc.url=jdbc:postgresql://<URL>:<Port>/<meinName>
```

### 2.1.3 Hibernate ORM mit Panache

Hibernate ORM ist der ORM für Quarkus. Da in Java objekt-orientiert programmiert wird und PostgreSQL eine relationale Datenbank ist, muss eine Möglichkeit gefunden werden, wie die in Java erstellten Objekte in die Datenbank übertragen werden. Da kommt ein ORM ins Spiel. Dieser "übersetzt" den Java-Code für die Datenbank. Dadurch

können Java-Klassen als Objekte persistiert werden und gelöscht werden, ohne dass komplexe Codezeilen konstruiert werden müssen. [6]

Panache bietet zusätzliche Klassen mit Funktionen, von denen abgeleitet werden kann. Diese erleichtern das Arbeiten mit angelegten Entitäten, besonders bei CRUD-Operationen. Zum Beispiel ist für das Persistieren eines neuen Objekts lediglich ein Aufruf der `.persist()`-Methode notwendig. [7]

### 2.1.4 REST-Easy

REST-Easy ist eine Erweiterung, die es ermöglicht, im Quarkus Projekt mit Jakarta RESTful Web Services zu arbeiten. Das heißt, dass durch diese Extension im Projekt APIs erstellt werden können. Im Code werden diese erstellt durch die zwei Annotationen `@Path` und `@<beliebige HTTP-Methode>`.

Folgende HTTP-Methoden werden in dieser Arbeit verwendet:

- GET
- POST
- DELETE

## 2.2 PostgreSQL

PostgreSQL ist ein open source Managementsystem für relationale Datenbanken, welches seit 35 Jahren entwickelt wird. Hinter diesem System befindet sich eine große Community, welche weiterhin Features veröffentlicht.

Die Entscheidung Eine gute Dokumentation und die vielen verschiedenen Anwendungsfälle [8]

## 2.3 IntelliJ IDEA

IntelliJ IDEA ist eine IDE, welche von JetBrains entwickelt wurde. Diese ist ausgelegt für Java- und Kotlin-Projekte und assistiert bei verschiedensten JVM-Frameworks. Durch eingebaute Features erleichtert diese Entwicklungsumgebung das Programmieren für Nutzende. Plug-Ins ermöglichen es, Datenbankverbindungen und weiteres in der

IDE zu konfigurieren, sodass dem\*der Entwickler\*in eine Übersicht von benötigten Informationen gegeben werden kann. [9]

## 2.4 LeoCloud

Die LeoCloud ist ein Projekt der HTL Leonding mit Aberger Christian als Projektleiter. Sie ermöglicht es mittels Kubernetes, eine beliebige Anwendung auf einer Cloud laufen zu lassen. Um diese Cloud nutzen zu können ist eine E-Mail-Adresse mit der HTL-Leonding-Schulsignatur notwendig. Da dies ein Schulinternes Projekt ist, war eine Kommunikation mit den Entwicklern gegeben, das heißt, dass potentielle Fragen sofort beantwortet werden konnten. [10]

## 2.5 Notion

Notion ist eine Art digitales Notizbuch, womit mit der so genannte *Workspaces* erstellt werden können. Diese ist verfügbar im Browser, als App in Windows-, MacOS- oder auf iOS- und Android-Geräten. Es besteht die Möglichkeit eine Seite zu erstellen und mit einem einzigen Befehl Unterseiten anzulegen, welche direkt verlinkt werden. Diese Seiten lassen sich mit beliebigem Inhalt füllen und durch einfache */-Befehle* ist es möglich z. B. ein Kanban Board erstellen. Es ist gut für Neueinsteiger, da alle Funktionalitäten gut beschrieben sind und es benutzerfreundlich gestaltet ist. Workspaces lassen sich mit anderen Nutzern teilen, wodurch jeder Nutzer auf eine *single source of truth* Version des Workspaces zugreifen kann. Für die Bearbeitung in echt-zeit wird Internet benötigt, falls dieses jedoch ausfällt, werden die Änderungen gespeichert und synchronisiert, sobald wieder eine Netzwerkverbindung aufgebaut wurde. [11]

Das Feature von geteilten Workspaces war für diese Arbeit nützlich, um Notizen nach Meetings und gemeinsame Termine festzuhalten, genauso um wichtige Links zu speichern und diese schnell wieder abrufen zu können. Unser Workspace war aufgeteilt in verschiedene Bereiche, wobei die wichtigsten Informationen auf der Titelseite gesichert werden. Siehe Abb. 1

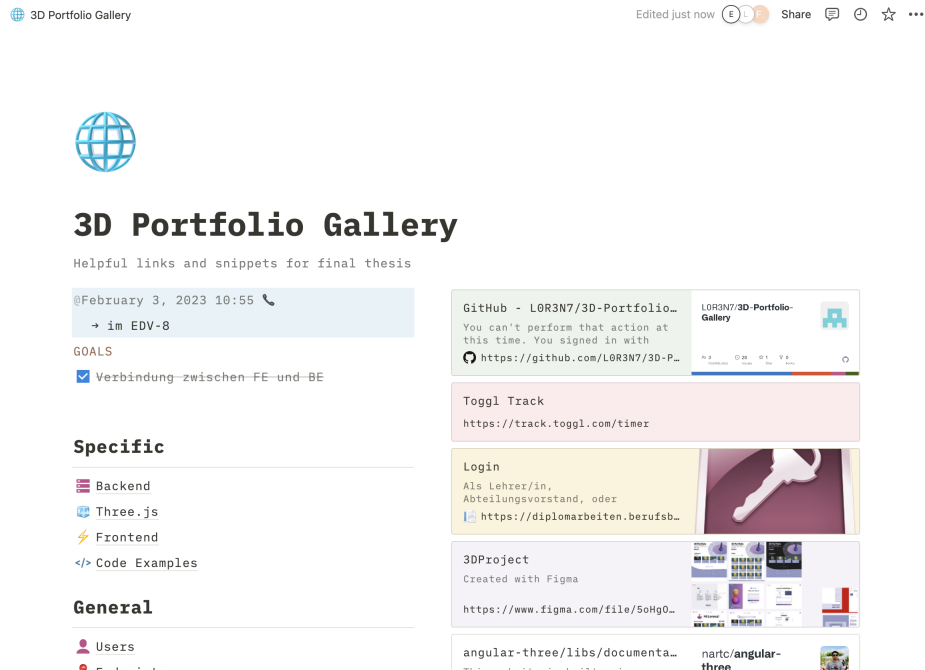


Abbildung 1: Notion Workspace der Diplomarbeit

## 2.6 PlantUML

PlantUML ist ein Tool, welches die Erstellung von UML-Diagrammen vereinfacht. Das Ziel von UML ist es, Tools zu liefern, um Systeme zu visualisieren. [12] Dafür gibt es verschiedene Arten von UML-Diagrammen, wie beispielsweise Objektdiagramme oder Sequenzdiagramme. [13]

Für diese Arbeit wurde ein ERD mit Krähenfußnotation, auch bekannt als Martin-Notation, verwendet.

## 2.7 JSON Web Token

JSON Web Token, oder JWT, sind eine Methode, um Daten sicher zu Übertragen. Sie können über eine digitale Signatur besitzen, wie zum Beispiel eines RSA Schlüsselpaars, oder eines HMAC Algorithmus. In diesem Token ist es möglich Daten zu speichern, sowie eine TTL festzulegen. Die Authentifizierung geschieht durch das Mitgeben des Tokens im Header. Jeder Token muss validiert werden, wodurch ihm eine Signatur hinzugefügt wird. Diese wird mittels den vorher genannten Methoden umgesetzt. [14]

# 3 Umsetzung

## 3.1 Design und Planung

### 3.1.1 Erstellung des ERDs

Um zu gewährleisten, dass die Systemanforderung getroffen werden, ist es notwendig ein entsprechendes ERD zu erstellen. Innerhalb eines ERD werden alle Entitäten, Attribute, sowie deren Beziehungen dargestellt. Es dient als gemeinsamer Nenner für die Projektentwicklung, da jedes Teammitglied eine eigene Vorstellung der Datenstruktur haben könnte. Ein weiterer Vorteil eines ERDs ist das Potenzial Fehlerquellen zu identifizieren und dadurch zukünftige Probleme zu umgehen.

Nachdem die benötigten Entitäten, sowie deren Attribute festgelegt wurden, müssen nun die Beziehungen zueinander festgelegt werden. Dabei wird gewählt zwischen One-to-One, One-to-Many und Many-to-Many Beziehungen, wobei die Many-to-Many Beziehung innerhalb einer Assoziationstabelle dargestellt wird.

Innerhalb dieser Arbeit ist die Entität *Exhibitions* das zentrale Objekt, was in der Abbildung 3 gut ersichtlich ist. Die Beziehungen zwischen *Positionen*, *Exhibits*, *Rooms* und *Exhibitions* stellten sich besonders als Herausforderung dar, aufgrund der vielen Abhängigkeiten zueinander. Diese wurden besonders erst in der Implementierung ersichtlich, wodurch das ERD im Laufe der Entwicklung etwas angepasst wurde. Weitere Gründe für die Abänderung, waren Planänderungen bei der Kommunikation zwischen zwei Entitäten, sowie der Aufbau der Hauptentitäten *Exhibition* und *Exhibit*. Die Unterschiede zwischen der ersten Version und der finalen sind deutlich (siehe Abb. 2 und 3). Besonders lassen sich die oben erläuterten Änderungen erkennen.

## 3.2 Aufsetzen der Datenbank und des Servers

Um das Quarkus-Projekt aufzusetzen, gibt es mehrere Möglichkeiten. Standardmäßig wird dazu die Webseite [code.quarkus.io](https://code.quarkus.io) verwendet, da hier die gewünschten Abhängigkeiten durch unkompliziertes Auswählen hinzugefügt werden können. Für diese Arbeit

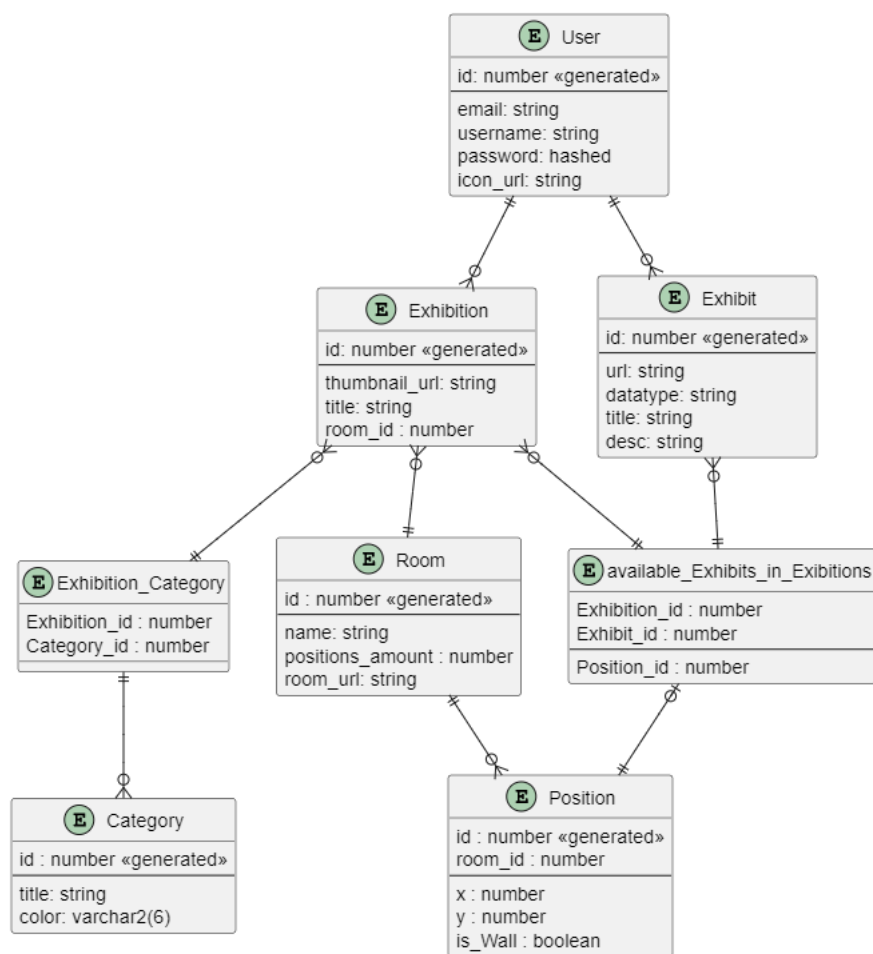


Abbildung 2: Erste Version des ERDs

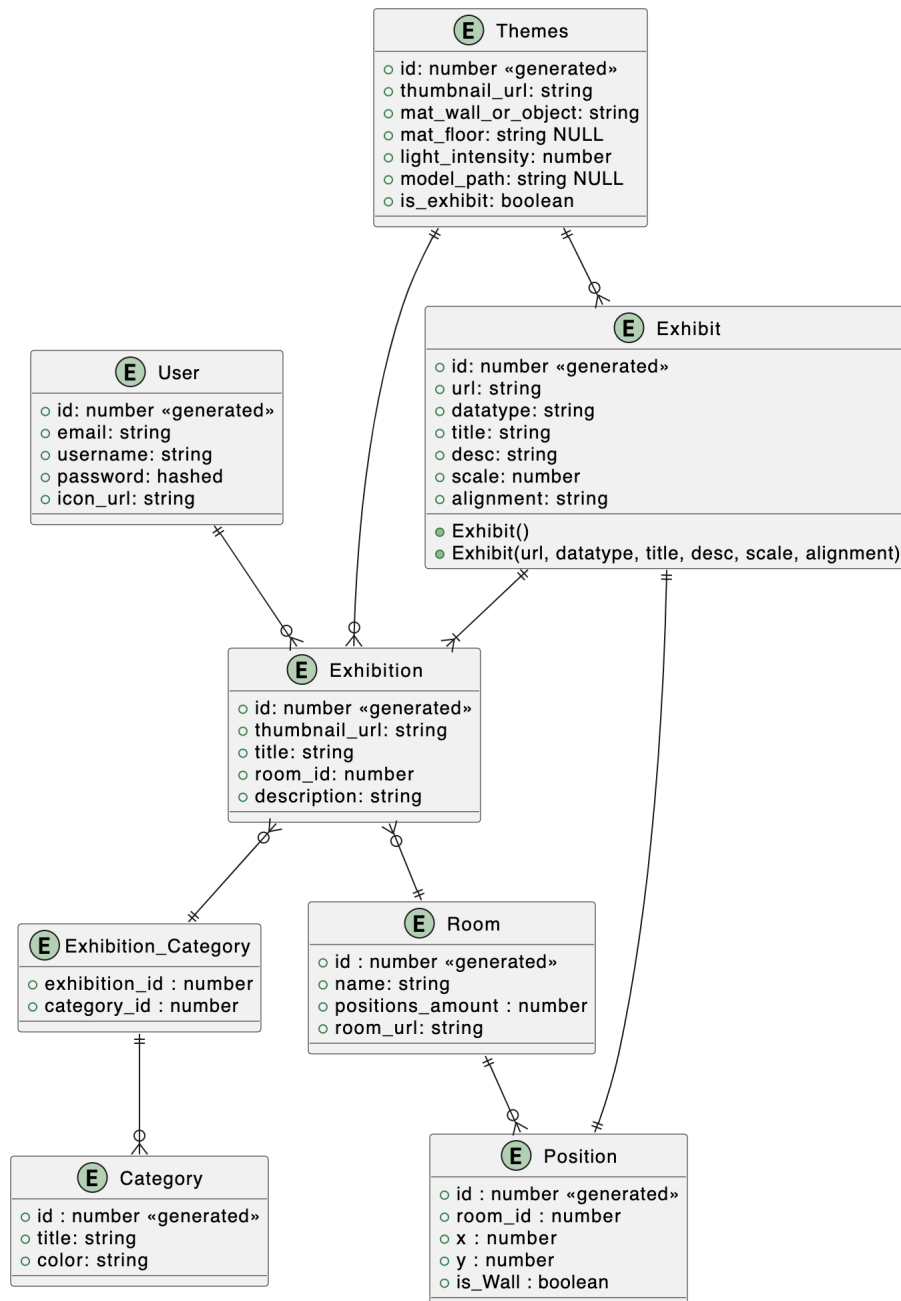


Abbildung 3: Finale Version des ERDs

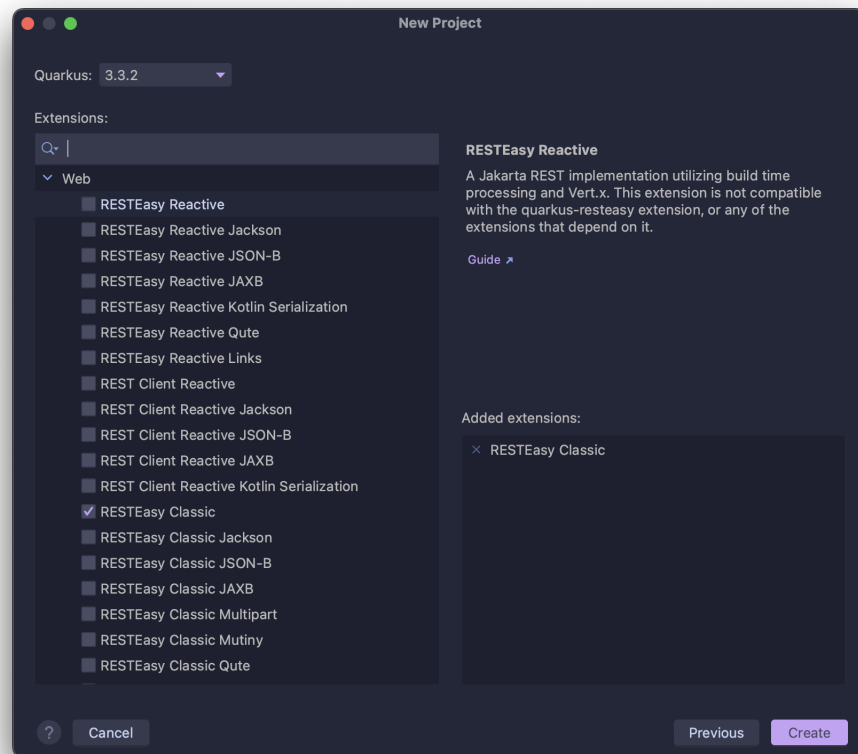


Abbildung 4: Auswählen der gewünschten Quarkus Extensions im Quarkus Plugin

wurde jedoch das IntelliJ Plug-in *Quarkus* von JetBrains s.r.o. verwendet. Dieses bildet dieselben Möglichkeiten, wie die Webseite, in einem eigenem Fenster ab (siehe Abb. 4). Zusätzlich werden Tools zur Verfügung gestellt, wie zum Beispiel Code-Assistenz, oder Laufkonfigurationen. [15]

Nach dem Aufsetzen des Projektes, wird eine Datenbankverbindung benötigt. Dazu wird PostgreSQL am Gerät installiert und gestartet. Name, Nutzer und Port der Datenbank können frei gewählt werden, allerdings wurden diese hier nach standardmäßiger Vorgabe gelassen.

Als nächsten Schritt wird eine Verbindung zu diesem lokalen Sysytem aufgebaut. Dafür werden die Einstellungen in Abb. 3.2 konfiguriert.

```

1      quarkus.datasource.db-kind=postgresql
2      quarkus.datasource.username = postgres
3      quarkus.datasource.password = postgres
4      quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/postgres
5
6      quarkus.hibernate-orm.database.generation=drop-and-create

```

Falls der Verbindungsaufbau versagt, wirft Quarkus während der Kompilierung Fehlermeldungen. Andernfalls können nun die Tabellen angelegt werden.



### 3.2.1 Umsetzung des ERDs in Quarkus

Um eine Tabelle zu erstellen wird, wie vorher erwähnt, die Quarkus-Abhängigkeit *Hibernate ORM mit Panache* verwendet. Mittels dieser wird eine Tabelle erstellt durch die Definition der Annotation `@Entity`. Üblicherweise benötigt jede Entitäten-Klasse eine definierte Id. Durch die Ableitung der *PanacheEntity*-Klasse (siehe Abb. 3.2.1, Zeile 2) kann jedoch auf diese verzichtet werden, da sie dort schon definiert wurde.

In der Klasse werden alle Attribute als Variablen, gemäß des ERDs definiert. Die Beziehungen zu anderen Entitäten werden durch die Annotationen `@OneToMany`, `@ManyToMany` und `@OneToOne` über der dazugehörigen Variable dargestellt. Auf der gegenseitigen Entität muss ebenso eine Variable erstellt werden, wobei da die Annotation etwas anders aussieht. Diese muss die *mappedBy*-Einstellung konfigurieren, da ansonsten die Beziehung mehrmals generiert wird (siehe Abb. 3.2.1, Zeile 4). Bei Many-to-Many wird eine zusätzliche Annotation `@JoinTable` hinzugefügt, um die Assoziationstabelle zu konfigurieren (siehe Abb. 3.2.1, Zeilen 5 bis 12). Zu den Relationen können zusätzlich noch Cascade Types festgelegt werden.

```

1  @Entity
2  public class Exhibition extends PanacheEntity {
3      // ...
4      @OneToMany(mappedBy = "exhibition", cascade = CascadeType.REMOVE)
5      public List<Exhibit> exhibits;
6
7      @ManyToMany
8      @JoinTable(
9          name = "exhibitions_categories",
10         joinColumns = @JoinColumn(name = "exhibition_id"),
11         inverseJoinColumns = @JoinColumn(name = "category_id")
12     )
13     public Set<Category> categories;
14 }
```

Für die Entitäten Theme, Positionen und Rooms hat sich das Team dafür entschieden, die Daten vorgefertigt in die Datenbank zu laden und die dazugehörigen Dateien manuell auf den Server zu speichern. Das bedeutet, dass für diese Entitäten keine Methoden angelegt werden müssen, um neue Objekte zu erstellen. Da in den *application.properties* die Art der Datenbank auf *drop-and-create* gesetzt wurde, war eine *import.sql*-Datei während der Entwicklung von großer Bedeutung. Diese lädt die vordefinierten Daten bei jedem Start automatisch in die Datenbank.

### 3.2.2 Implementierung der REST-Schnittstellen

Um das Frontend auf die Daten zugreifen zu lassen werden Schnittstellen benötigt, die dies ermöglichen. Dafür werden im Backends Resource-Klassen erstellt, um Endpoints zu verwalten. Diese definiert man durch die `@Path`-Annotation, da diese angibt, unter

welchem Pfad die Methoden erreichbar sind. Für jede Methoden muss zusätzlich die HTTP-Methode als Annotation definiert werden.

## Repositories

Um in Resource-Klassen die Datenbank anzusprechen, müssen die benötigten Repositories injiziert werden. Durch die Verwendung der Panache Extension, implementieren alle Repository-Klassen das *PanacheRepository*, wobei die zugehörige Tabelle in spitzen Klammern angegeben werden muss, wie in Code Beispiel 2 bei Zeile 2. Dieses fügt einfache Methoden für CRUD-Operationen hinzu, ohne diese zusätzlich zu definieren.

Listing 2: Teil aus dem Exhibition Repository

```

1  @ApplicationScoped
2  public class ExhibitionRepo implements PanacheRepository<Exhibition> {
3      //...
4      public List<ExhibitionWithUserRecord> listAllExhibitionsWithUserField() {
5          String sql = "select new
6              org.threeDPortfolioGallery.records.ExhibitionWithUserRecord(e,
6                  u.user_name, u.icon_url) from Exhibition e join e.user u left join
6                  e.categories c";
7          TypedQuery<ExhibitionWithUserRecord> q = getEntityManager()
8              .createQuery(sql, ExhibitionWithUserRecord.class);
9          var ret = q.getResultList();
10         return ret;
11     }
12 }
```

Die Repositories ermöglichen ebenso das Definieren von eigenen JPQL-Befehlen. JPQL ermöglicht, neben der Nutzung der gewohnten SQL-Keywods, das definieren von Parametern in Queries, sowie die Verwendung von eigenen Records und DTOs.

## Records und DTOs

Records sind eine Möglichkeit, einen Datensatz zurückzugeben, mit Werten aus verschiedenen Tabellen. Sie können gesetzt werden, jedoch nicht mehr geändert werden. Deswegen eignen sie sich besonders gut, Daten aus SQL-Statements zu speichern und zu returnen.

DTOs hingegen können mittels Set-Methoden die Werte wieder ändern. Dies ist nicht unbedingt notwendig beim Auslesen der Datenbank, aber sehr nützlich beim Anlegen neuer Objekte.

## User

Ein User wird angelegt durch das Definieren eines Nutzernamens, sowie eines Passwortes. Optional können ebenso Pfade angegeben werden für Profilbilder. Jedes Passwort

bekommt vor dem Speichervorgang einen Salt angehängt und wird mithilfe von Base64 verschlüsselt.

Der Zugriff auf die Schnittstelle ist gesichert durch ein Tokensystem. Nach jedem erfolgreichem login wird ein Token zurückgesendet, der für die Authentifizierung genutzt werden kann. In diesem Token ist der Nutzernamen gespeichert, sowie zu welcher Gruppe der Nutzer angehört. Je nach Gruppe kann auf verschiedene Endpoints zugegriffen werden. Die Limitationen werden bei der Definition der Methoden angegeben mittels `@PermitAll`, was keine Limitationen hinzugefügt. Genauso gibt es jedoch `@RolesAllowed`, was nur definierten Rollen den Zugriff erlaubt.

## Exhibitions

Eine der wichtigsten Funktionen der Exhibitions-Resource, ist das Anlegen einer Exhibition. Dafür wird eine `@POST`-Methode erstellt, die ein Objekt nach dem DTO in Codeausschnitt 3 einliest. DTOs können auch verschachtelt werden, was in Zeile 12 erkennbar ist.

In der Klasse wurde ebenfalls die Annotation `@Data` verwendet. Diese stammt aus der Java Library Project Lombok und automatisiert die Erstellung von Get- und Set-Methoden. [16]

Listing 3: Exhibition DTO

```
1      @Data
2      public class AddExhibitionDTO {
3          String thumbnail_url;
4          String title;
5          String description;
6
7          Long room_id;
8
9          Long user_id;
10         Long[] category_ids;
11
12         AddExhibitDTO[] exhibits;
13     }
```

Da die Methode ein DTO erwartet, muss definiert werden, dass etwas eingelesen werden soll. Dies macht man durch das `@Consumes`. Der Codeabschnitt 4 Zeile 4 legt fest, dass beim Senden der POST-Methode ein JSON-Objekt erwartet werden soll. Der Ausdruck in der Klammer in Zeile 5 im selben Abschnitt beschreibt, wie dieses Objekt konstruiert sein soll.

Beim Aufruf der Methode wird das mitgegebene Objekt in der Variable `newExhibition` gespeichert. Danach werden zwei leere Variable initialisiert. Zum einen eine neue Exhibition, zum anderen eine LinkedList bestehend aus Exhibits. Diese sind getrennt, da

alle Exhibits zuerst in die Datenbank eingefügt werden müssen, bevor die Exhibition tatsächlich gespeichert wird.

Neben den Exhibits, müssen genauso die anderen Werte der mitgegebenen Variable geprüft werden. Das heißt, es muss sichergestellt werden, dass keine nicht-existierenden User- oder Category-Ids übergeben wurden. Falls etwas einen Fehler während der Überprüfung wirft, Fehlermeldung zurückgegeben mit dem Statuscode 406 "Not Acceptable". Gleichzeitig geschieht ein Rollback der davor gespeicherten Objekte. Diese Vorgehensweise wird durch die Annotation *@Transactional* definiert.

#### Listing 4: Methode zum Anlegen von Exhibitions

```
1  @POST
2  @Transactional
3  @Path("/new")
4  @Consumes(MediaType.APPLICATION_JSON)
5  public Response postNewExhibition(AddExhibitionDTO newExhibition){
6      Exhibition exhibition = new Exhibition();
7      List<Exhibit> newExhibitList = new LinkedList<>();
8      // ...
9  }
```

## Filemanagement

Der File-Upload geschieht während der Bearbeitung der Exhibition im Frontend. Nach jeder Dateiauswahl des Nutzers der Webseite, wird der Endpoint zuständig für den Fileupload aufgerufen. Dieser erwartet sich, anders als die anderen POST-Methoden, kein JSON-Objekt, sondern eine Datei. Wiedermals wird dies festgelegt mittels der *@Consumes*-Annotation, mit *"multipart/form-data"* als Wert. Das Limit der akzeptierten Dateien wird in den *application.properties* (Abb. 3.2.2) festgelegt.

```
1  quarkus.http.limits.max-body-size = 300m
```

Für das eigentliche Abspeichern einer Datei, ist die Methode *writeFile()* zuständig (siehe Abb. 5). Diese wird aufgerufen, wenn der Endpoint aufgerufen wird und eine Datei mitgegeben wurde. Der erste Parameter der Methode, das Byte-Array, wird mittels eines Inputstreams erstellt. Der Filename wird in einer extra Methode erstellt. Dafür wird der ursprüngliche Name der Datei genommen und untersucht auf Leerzeichen. Sobald diese entfernt wurden, wird der abgeänderte Name zurückgegeben.

Nachdem das File abgespeichert wurde, wird dessen Pfad mit dem neuen Namen in die Response mitgegeben. Diese Antwort wird dann als Wert erwartet als Url, beim Anlegen neuer Exhibits.

#### Listing 5: Hochladen der Dateien

```
1     private void writeFile(byte[] content, String filename) throws IOException {
2         File file = new File(filename);
3         if (!file.exists()) {
4             file.createNewFile();
5         }
6         FileOutputStream fos = new FileOutputStream(file);
7         fos.write(content);
8         fos.flush();
9         fos.close();
10    }
```

## 3.3 Hosten auf einer Cloud

## 3.4 Schreiben der Tests

next termin: fertig mit praktischem dienstag 6. EH (12:45) am 3.10 treffen beim eingang  
sonntag abend immer schicken

feedback: - weglassen von der anfangsfrage - umsetzung gliedern nach user stories

## **4 Zusammenfassung**

# Glossar

**API** Der Begriff steht für *Application Programming Interface*. Durch APIs können verschieden Applikationen miteinander kommunizieren, ohne die jeweils andere Implementierung wissen zu müssen. [17]

**Cascade Types** Es gibt verschiedene Arten von Cascade Types. Sie definieren, was mit Abhängigkeiten passiert, wenn diese bearbeitet oder gelöscht werden. [18]

**CLI** Command Line Interface

**CRUD** Create Replace Update Delete

**DTO** Data Transfer Object

**ERD** Der Begriff steht für *Entity Relationship Diagram*. Diese Diagramme visualisieren, welche Beziehungen verschiedenen *Entitäten* zueinander haben. [19]

**Framework** Frameworks bieten Funktionen an, ohne dass diese selbst implementiert werden müssen. Sie erleichtern das Programmieren in norm. [20]

**HTTP** Hypertext Transfer Protocol

**IDE** Integrated Development Environment

**JDBC** Java Database Connectivity

**JPQL** Jakarta Persistence Query Language

**JSON** JavaScript Object Notation

**JVM** Java Virtual Machine

**JWT** JSON Web Token

**ORM** Object Relational Mapper

**REST** Representational State Transfer

**SQL** Structured Query Language

**TTL** Time To Live

**UML** Unified Modeling Language



# Literaturverzeichnis

- [1] Quarkus, „SUPERSONIC/SUBATOMIC/JAVA,” WebPage, 2023, letzter Zugriff am 03.09.2023. Online verfügbar: <https://quarkus.io>
- [2] —, „What is Quarkus?” WebPage, 2023, letzter Zugriff am 01.09.2023. Online verfügbar: <https://quarkus.io/about/>
- [3] —, „What is Quarkus?” WebPage, 2023, letzter Zugriff am 01.09.2023. Online verfügbar: <https://quarkus.io/get-started/>
- [4] T. A. S. Foundation, „PostgreSQL: about,” WebPage, 2023, letzter Zugriff am 02.09.2023. Online verfügbar: <https://maven.apache.org/what-is-maven.html>
- [5] Mureinik, „What is the difference between JDBC API and PostgreSQL Driver?” WebPage, 2020, letzter Zugriff am 02.09.2023. Online verfügbar: <https://stackoverflow.com/questions/63678556/what-is-the-difference-between-jdbc-api-and-postgresql-driver>
- [6] G. Chehab, „What Is an ORM? How Does It Work? How Should We Use One?” WebPage, 2023, letzter Zugriff am 03.09.2023. Online verfügbar: <https://www.baeldung.com/cs/object-relational-mapping>
- [7] Quarkus, „SIMPLIFIED HIBERNATE ORM WITH PANACHE,” WebPage, 2023, letzter Zugriff am 07.09.2023. Online verfügbar: <https://quarkus.io/guides/hibernate-orm-panache>
- [8] T. P. G. D. Grou, „What is Maven?” WebPage, 2023, letzter Zugriff am 02.09.2023. Online verfügbar: <https://www.postgresql.org/about/>
- [9] JetBrains, „Features overview,” WebPage, 2023, letzter Zugriff am 02.09.2023. Online verfügbar: <https://www.jetbrains.com/idea/features/>
- [10] A. Christian, „LeoCloud User Manual,” WebPage, 2021, letzter Zugriff am 03.09.2023. Online verfügbar: [https://cloud.htl-leonding.ac.at/user-manual.html?#\\_allgemeines](https://cloud.htl-leonding.ac.at/user-manual.html?#_allgemeines)
- [11] N. Labs, „Über Notion,” WebPage, 2023, letzter Zugriff am 03.09.2023. Online verfügbar: <https://www.notion.so/de-de/about>
- [12] O. Object Management Group, „OMG® Unified Modeling Language® (OMG UML®),” WebPage, 2017, letzter Zugriff am 03.09.2023. Online verfügbar: <https://www.omg.org/spec/UML/2.5.1/PDF>
- [13] PlantUML.com, „PlantUML – Ein kurzer Überblick,” WebPage, letzter Zugriff am 03.09.2023. Online verfügbar: <https://plantuml.com/de/>
- [14] Auth0, „What is JSON Web Token?” WebPage, 2023, letzter Zugriff am 09.09.2023. Online verfügbar: <https://jwt.io/introduction>

- [15] J. s.r.o., „Quarkus - IntelliJ IDEs Plugin | Marketplace,” WebPage, 2023, letzter Zugriff am 26.10.2023. Online verfügbar: <https://plugins.jetbrains.com/plugin/20306-quarkus>
- [16] T. P. L. Authors, „Project Lombok,” WebPage, 2023, letzter Zugriff am 09.09.2023. Online verfügbar: <https://projectlombok.org/#>
- [17] R. Hat, „What is an API?” WebPage, 2022, letzter Zugriff am 21.08.2023. Online verfügbar: <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>
- [18] baeldung, „Overview of JPA/Hibernate Cascade Types,” WebPage, 2021, letzter Zugriff am 07.09.2023. Online verfügbar: <https://www.baeldung.com/jpa-cascade-types>
- [19] V. Paradigm, „What is Entity Relationship Diagram (ERD)?” WebPage, 2023, letzter Zugriff am 24.08.2023. Online verfügbar: <https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/;WWWSESSIONID=617AD2BCF587341A07E1E81EA0E2099C.www1>
- [20] C. Team, „What is a Framework?” WebPage, 2021, letzter Zugriff am 07.09.2023. Online verfügbar: <https://www.codecademy.com/resources/blog/what-is-a-framework/>

# Abbildungsverzeichnis

1	Notion Workspace der Diplomarbeit . . . . .	6
2	Erste Version des ERDs . . . . .	8
3	Finale Version des ERDs . . . . .	9
4	Auswählen der gewünschten Quarkus Extensions im Quarkus Plugin . .	10

# Tabellenverzeichnis

# Quellcodeverzeichnis

1	Beispielkonfigurationen . . . . .	3
2	Teil aus dem Exhibition Repository . . . . .	12
3	Exhibition DTO . . . . .	13
4	Methode zum Anlegen von Exhibitions . . . . .	14
5	Hochladen der Dateien . . . . .	14

# Anhang